Two PhD studentships available at Imperial College London on Automatic Verification and Synthesis of Device Drivers

The Multicore Programming Group in the Department of Computing at Imperial College London are looking to recruit two PhD students to work on an exciting new project, “Automatic Synthesis of High-Assurance Device Drivers”. The project is funded by a gift from Intel Corporation, and is a collaboration between Imperial, NICTA (Australia), University of Colorado Boulder (USA) and University of Toronto (Canada).

Each position is for three years, and the positions are fully funded for candidates whose country of origin is within the European Union. The start dates for these positions are flexible, until October 2013. The closing date for applications is:

30 April 2013

The broad topics of the studentships are as follows (though there is room for flexibility and overlap):

- Programming language support and verification methods to aid in the development of complex, concurrent device drivers
- Automatic proof generation techniques for device driver synthesis

Further details about the project, topics and desired skill sets are provided below.

The successful candidates will join the Multicore Programming Group at Imperial, led by Dr Alastair Donaldson. The Department of Computing at Imperial is among the strongest in the world, and London is a vibrant and exciting city in which to be based. The project will feature a strong industrial collaboration with Intel Corporation, and the team at Imperial will interact frequently with the teams at NICTA, Boulder and Toronto.

If you are interested in applying for one of these positions then please contact the Principal Investigator, Alastair Donaldson (alastair.donaldson@imperial.ac.uk).

Background and aims of the project:

Device drivers are hard to develop and are notoriously unreliable. While constant innovation in the area of electronic design automation has led to dramatic improvements in the IC design process, device driver development practices have not changed much since the days of mainframe computers. As a result, it is common nowadays for product releases to be determined by driver development and validation schedules rather than those of silicon.

To address this long-standing problem, we propose a new driver development methodology that will allow faster creation of device drivers with fewer defects. The innovation at the heart of our methodology is the automatic synthesis of correct-by-construction device drivers from a formal model of the hardware device and a specification of the driver/OS interface. For complex, concurrent device drivers for which full synthesis is not possible, we plan to investigate programming language support and automated verification technology to aid in the manual construction of reliable drivers. Here the goal is to achieve full functional verification of driver behaviour, not simply to find bugs in the driver implementation.

PhD topics in the Imperial team

The above is a brief overview of the whole project. At Imperial, we are looking for a PhD student to work on each of the following areas. However, as noted above, there is scope for flexibility in the PhD topics undertaken, within the overall aims of the project.

Programming language support and verification methods to aid in the development of complex, concurrent device drivers: A PhD student working on this topic will investigate language and tool support for writing verifiable drivers. This could, for example, involve identifying a restricted subset of C that allows full driver functionality to be expressed, but which features a conservative type system, limited pointer arithmetic (which together guarantee type safety), and specially managed dynamic memory allocation. These restrictions will allow the verifier to make stronger assumptions when reasoning about access to shared data, simplifying the process of reasoning about correct concurrency. Verification could be using theorem provers (e.g., building on the success of the GPUVerify project at Imperial), or through model checking. As well as developing new techniques that are theoretically sound, there will be an emphasis on building prototype tools that realise these techniques, which can be used by the other project partners. For instance, counterexamples generated by the verification technique for concurrent programs can be used as input to the synthesis algorithm to allow automatic placement of inter-thread synchronization primitives.

Automatic proof generation techniques for device driver synthesis: During the project, and led by other project partners, we will develop novel synthesis techniques for generating device driver implementations that are correct by construction. As well as collaborating on this effort, a PhD student in the Imperial team will investigate a proof generation approach whereby the synthesis technique will generate both a driver implementation and an associated proof of correctness. This proof will be produced by capturing decisions made by the synthesis algorithm using formal logic. Capturing this proof as a script to be checked by a theorem prover such as Isabelle or Coq will yield a very high degree of assurance that the synthesized driver is correct, as well as flagging up incorrect proofs arising due to bugs in the synthesis algorithm implementation.

Desired skill set

We are looking for candidates with a strong grounding in Computer Science or a related discipline, holding a Master’s degree or equivalent, and with a mixture of theoretical and practical skills. Because the project has a practical focus, is relevant to system-level software, requires rigorous reasoning, and involves multiple partners, the following are essential:

- Strong programming skills, and an interest in low-level programming
- Basic knowledge of operating systems and compilers
- Logical reasoning skills
- A keenness to collaborate with other academic and industrial partners

We are particularly keen to recruit a candidate who has some experience in a subset of the following areas (we do *not* expect to recruit a candidate who has experience in all these areas!):

- System-level implementation (e.g., writing device drivers, or hacking on the Linux kernel)
- Programming language implementation (e.g., compiler-writing, or building static analysers)
- Formal verification techniques such as model checking or theorem proving
- Concurrent programming
- Using mechanical theorem provers such as Coq or Isabelle